Army Research Laboratory

Basic Searching, Interpolating, and Curve-Fitting Algorithms in C++

by Robert J Yager

ARL-TN-0657 January 2015

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5066

ARL-TN-0657 **January 2015**

Basic Searching, Interpolating, and Curve-Fitting Algorithms in C++

Robert J Yager
Weapons and Materials Research Directorate, ARL

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**			
1. REPORT DATE *(DD-MM-YYYY)* January 2015	**2. REPORT TYPE** Final		**3. DATES COVERED (From - To)** January 2014–July 2014
4. TITLE AND SUBTITLE Basic Searching, Interpolating, and Curve-Fitting Algorithms in C++			**5a. CONTRACT NUMBER**
			5b. GRANT NUMBER
			5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Robert J Yager			**5d. PROJECT NUMBER** AH80
			5e. TASK NUMBER
			5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) US Army Research Laboratory ATTN: RDRL-WML-A Aberdeen Proving Ground, MD 21005-5066			**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TN-0657
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			**10. SPONSOR/MONITOR'S ACRONYM(S)**
			11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			
13. SUPPLEMENTARY NOTES			
14. ABSTRACT This report documents a set of functions, written in C++, that can be used to perform interpolations (nearest-neighbor, linear, and cubic) and to find coefficients for best-fit equations. Functions for working with periodic equations are included.			
15. SUBJECT TERMS interpolate, linear, cubic, hermite, polynomial, fit			

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Robert J Yager
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	UU	40	**19b. TELEPHONE NUMBER** (*Include area code*) 410-278-6689

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

Acknowledgments

I would like to thank Tim Fargus of the US Army Armament Research, Development, and Engineering Center's System Analysis Division. Tim provided technical and editorial recommendations that improved the quality of this report.

I would also like to thank Dr Douglas Chapman, Professor of Chemistry at Southern Oregon University, for the concepts he taught me in his "Computer Applications in Chemistry" class. Dr Chapman taught me that it's possible to use very simple computer code to solve complex problems. At the time, I didn't realize how valuable that lesson was.

INTENTIONALLY LEFT BLANK.

1. Introduction

This report documents a set of functions, written in C++, that can be used to perform interpolations (nearest-neighbor, linear, and cubic) and to find coefficients for best-fit equations. Functions for working with periodic equations are included.

Measurements can be taken and stored using a variety of schemes. The interpolations described in this report are designed to work with data sets where measured values for all independent and dependent variables are explicitly stored. Furthermore, for data that has more than 1 independent variable, the measurements must all lie on some type of rectangular grid, where all grid locations are populated.

When measurements do not lie on some type of rectangular grid, interpolations become more difficult. For those types of data sets, KD-trees can be used to perform efficient nearest-neighbor searches, which can act as interpolations. The yKDTree namespace[1] can be used to work with KD-trees using C++.

The yBilinear namespace[2] can be used for the special case of data sets with 2 independent variables, where all measurements lie on a rectangular grid, all grid locations are populated, and all grid lines are evenly spaced.

The functions that are described in this report have been grouped into the yInterp namespace, which is summarized at the end of this report. The yInterp namespace relies exclusively on standard C++ operations and functions. However, example code that is included in this report makes use of the yRandom namespace[3] for generating pseudorandom numbers, the yBmp namespace[4] for creating images, and the yIo2[5] namespace for reading and parsing text files.

2. Background

My motivation for this project began with a task that involved interpolating a look-up table with 8 independent variables, some of which were periodic. Prior to this project, I had always written interpolators on an ad hoc basis. However, the thought of having to deal with an array that requires 8 indices to access a particular element (something like A[i][j][k][l][m][n][o][p]) convinced me that I needed a more formal set of tools.

The task required that all code be written in C++. To maximize portability, the code had to be written without the use of any platform-specific or C++11 tools. Performance was a priority. In addition, I wanted a versatile set of tools that I could use for future projects. The tools needed to be simple to use and well-tested so that I could feel comfortable sharing them with other programmers.

3. Searching Sorted Arrays: The BinarySearch() Function

The BinarySearch() function uses a binary search algorithm to search arrays that are sorted into ascending order. For arrays that do not contain duplicate entries, the BinarySearch() function can be used to find a pointer **c** such that

$$*\mathbf{c} \le \mathbf{k} < *(\mathbf{c}+1), \tag{1}$$

where **k** is key on which the search is based.

For arrays that contain duplicate entries, Eq. 1 needs to be slightly modified:

$$*\mathbf{c} \le \mathbf{k} \le *(\mathbf{c}+1). \tag{2}$$

Thus, if **k** is equal to an element that has one or more duplicates, **c** may point to any one of the duplicates.

Performance for the BinarySearch() function is $O(\log(n))$, which is demonstrated in Section 3.7.

3.1 BinarySearch() Code

```
template<class T>T*BinarySearch(//<======FIND THE POINTER c | *c <= k < *(c+1)
    T*a,T*b,//<--ARRAY START & END POINTS (ARRAY MUST BE SORTED IN INC. ORDER)
    T k){//<---------------------------------------------------------------KEY
  if(k<*a)return a-1;//..........note that a-1 may point to an invalid address
  for(T*c;k<*--b;k>*c?a=c:b=c+1)c=a+(b-a)/2;/*->*/return b;
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```

3.2 BinarySearch() Template Class

T	**T** must be a sortable data type.

3.3 BinarySearch() Parameters

a	**a** points to the beginning of the array that will be searched. The array should be sorted into ascending order.
b	**b** points to one element past the end of the array that will be searched. Thus, **b** is used to define the end of the search region but is not included in the search. **b** should be greater than **a**.
k	**k** is the key on which the search is based.

3.4 BinarySearch() Return Value

For arrays that do not contain duplicate entries, the BinarySearch() function returns a pointer to the greatest array element that is less than or equal to **k**. If **k** is less than the least array element, then **a**-1 is returned. Care needs to be taken when accessing the return pointer, since **a**-1 may not point to a valid address.

3.5 BinarySearch() Simple Example

The following example uses the BinarySearch() function to search a small sorted array that contains duplicates.

```
#include <cstdio>//....................................................printf()
#include "y_interp.h"//...................................................yInterp
int main(){//<=========A SIMPLE EXAMPLE FOR THE yInterp::BinarySearch() FUNCTION
  int X[11]={30,40,41,42,42,42,42,43,44,45,50};
  printf("X={");/*&*/for(int i=0;i<11;++i)printf("%d%s",X[i],i!=10?",":"}\n\n");
  printf(" *c | KEY | *(c+1) | INDEX\n---------------------------\n");
  for(int k=28;k<52;++k){
    int*c=yInterp::BinarySearch(X,X+11,k);
    if(c<X)printf(" -- |%4d |%5d   |%4d\n",k,*(c+1),c-X);
    else if(c==X+10)printf("%3d |%4d |   --   |%4d\n",*c,k,c-X);
    else printf("%3d |%4d |%5d   |%4d\n",*c,k,*(c+1),c-X);}
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```

OUTPUT:

```
X={30,40,41,42,42,42,42,43,44,45,50}

 *c | KEY | *(c+1) | INDEX
---------------------------
 -- |  28 |   30   |  -1
 -- |  29 |   30   |  -1
 30 |  30 |   40   |   0
 30 |  31 |   40   |   0
 30 |  32 |   40   |   0
 30 |  33 |   40   |   0
 30 |  34 |   40   |   0
 30 |  35 |   40   |   0
 30 |  36 |   40   |   0
 30 |  37 |   40   |   0
 30 |  38 |   40   |   0
 30 |  39 |   40   |   0
 40 |  40 |   41   |   1
 41 |  41 |   42   |   2
 42 |  42 |   42   |   5
 43 |  43 |   44   |   7
 44 |  44 |   45   |   8
 45 |  45 |   50   |   9
 45 |  46 |   50   |   9
 45 |  47 |   50   |   9
 45 |  48 |   50   |   9
 45 |  49 |   50   |   9
 50 |  50 |   --   |  10
 50 |  51 |   --   |  10
```

3.6 BinarySearch() Text Example

The following example uses the BinarySearch() function to determine where a word should be inserted into a sorted list.

```
#include <iostream>//.....................................................cout
#include <string>//.....................................................string
#include "y_interp.h"//...................................................yInterp
int main(){//<===========A TEXT EXAMPLE FOR THE yInterp::BinarySearch() FUNCTION
  std::string S[6]={"defective","defend","defendant","defender","defense",
    "defensive"},s="defenestrate";
  for(int i=0;i<6;++i)std::cout<<S[i]<<(i==5?"\n\n":", ");
  std::string*c=yInterp::BinarySearch(S,S+6,s);
  std::cout<<"The word "<<s<<" should be between "<<*c<<" and "<<*(c+1)<<".\n";
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```

OUTPUT:

```
defective, defend, defendant, defender, defense, defensive

The word defenestrate should be between defender and defense.
```

3.7 BinarySearch() Performance

The following example begins by using the yRandom namespace to populate an array with 2^{14} pseudorandom numbers using the Mersenne twister 19937 algorithm. The array is then sorted using the sort() function. Average search times are calculated for searches performed using a linear search algorithm and the BinarySearch() function. Figure 1 presents a graph of the results.

The average search time for the LinearSearch() function is $O(n)$. The average search time for the BinarySearch() function is $O(\log(n))$.

```
#include <algorithm>//....................................................sort()
#include <cstdio>//.....................................................printf()
#include <ctime>//........................................clock(),CLOCKS_PER_SEC
#include "y_interp.h"//..................................................yInterp
#include "y_random.h"//..................................................yRandom
template<class T>T*LinearSearch(//<========FIND THE POINTER c | *c <= k < *(c+1)
    T*a,T*b,//<----ARRAY START & END POINTS (ARRAY MUST BE SORTED IN INC. ORDER)
    T k){//<-----------------------------------------------------------------KEY
  if(k<*a)return a-1;//...........note that a-1 may point to an invalid address
  while(k<*--b);/*->*/return b;
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
int main(){//<=========PERFORMANCE TEST FOR THE yInterp::BinarySearch() FUNCTION
  const int N=16384,M=10000000;//...max # of elements in array, # of repetitions
  unsigned I[625];/*<-*/yRandom::Initialize(I,1);//....state of Mersenne twister
  double*X=new double[N];/*<-*/for(int i=0;i<N;++i)X[i]=yRandom::RandU(I,0,N);
  std::sort(X,X+N);//.......X is a sorted, unchanging list of numbers to search
  printf("          |     linear  search     |     binary  search\n");
  printf("    size  |------------------------|------------------------\n");
  printf("     of   |  search  |    index     |  search  |    index\n");
  printf("   array  | time (s) |   average    | time (s) |   average\n");
  printf(" ---------|----------|-------------|----------|-------------\n");
  for(int m=2;m<16385;m*=2){
    printf("%8d  |",m),yRandom::Initialize(I,1);
    double y=0,z=0,t=clock();
    for(int i=0;i<M;++i)y+=LinearSearch(X,X+m,yRandom::RandU(I,0,m))-X;
    printf("%8.3f  |%12.6f |",(clock()-t)/CLOCKS_PER_SEC,y/M),t=clock();
    yRandom::Initialize(I,1);
    for(int i=0;i<M;++i)z+=yInterp::BinarySearch(X,X+m,yRandom::RandU(I,0,m))-X;
    printf("%8.3f  |%12.6f\n",(clock()-t)/CLOCKS_PER_SEC,z/M);}
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```

OUTPUT:

```
          |     linear  search     |     binary  search
    size  |------------------------|------------------------
     of   |  search  |    index     |  search  |    index
   array  | time (s) |   average    | time (s) |   average
 ---------|----------|-------------|----------|-------------
       2  |   0.140  |   -0.731079 |   0.125  |   -0.731079
       4  |   0.140  |    0.555495 |   0.156  |    0.555495
       8  |   0.203  |    1.833760 |   0.224  |    1.833760
      16  |   0.244  |    5.133788 |   0.234  |    5.133788
      32  |   0.312  |   13.015384 |   0.327  |   13.015384
      64  |   0.390  |   31.287311 |   0.359  |   31.287311
     128  |   0.531  |   66.159459 |   0.421  |   66.159459
     256  |   0.936  |  128.260683 |   0.530  |  128.260683
     512  |   1.545  |  258.242078 |   0.592  |  258.242078
    1024  |   2.902  |  519.408822 |   0.671  |  519.408822
    2048  |   5.678  | 1035.587832 |   0.733  | 1035.587832
    4096  |  11.217  | 2054.054591 |   0.795  | 2054.054591
    8192  |  22.142  | 4091.340401 |   0.873  | 4091.340401
   16384  |  43.945  | 8193.415153 |   0.968  | 8193.415153
```

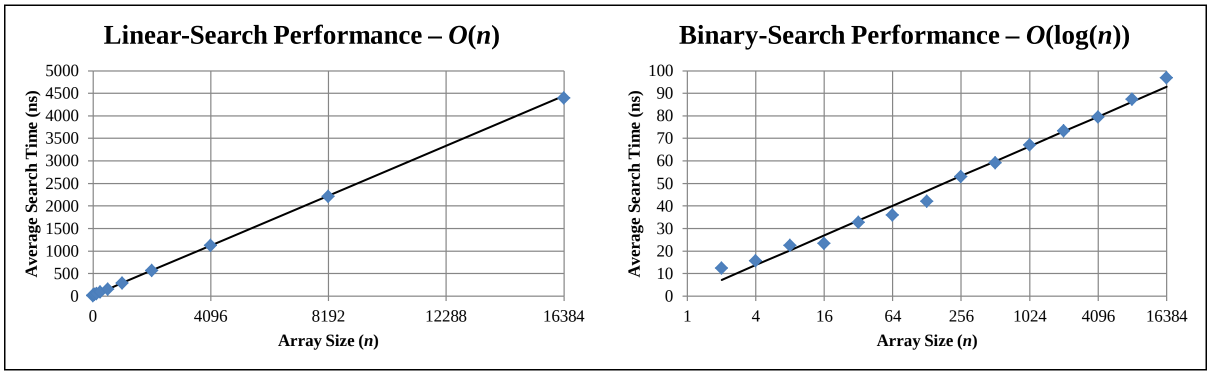


Fig. 1 Performance for linear and binary searches (right graph is semi-log)

4. Searching Sorted Arrays: The PeriodicSearch() Function

The PeriodicSearch() function uses the BinarySearch() function to search arrays that are associated with periodic functions, such as f, where

$$f(x+np)=f(x)\,. \tag{3}$$

Thus, the PeriodicSearch() function can be used to find a pointer **c** such that

$$*\mathbf{c} \le \mathbf{k} + n\mathbf{p} < *(\mathbf{c}+1), \tag{4}$$

where **k** and **p** are user supplied values and the integer n is chosen such that

$$*\mathbf{a} \le \mathbf{k} + n\mathbf{p} < *\mathbf{a}+\mathbf{p}. \tag{5}$$

4.1 PeriodicSearch() Code

```
template<class T>T*PeriodicSearch(//<=======FOR ARRAYS WITH PERIODIC VARIABLES
    T*a,T*b,//<--STARTING & ENDING POINTS (ARRAY MUST BE SORTED IN INC. ORDER)
    T&k,//<------------------------------------KEY (WILL BE SET TO k'=k+np)
    T p){//<------------------------------------------------------PERIOD (p>0)
  return BinarySearch(a,b,k=fmod(k-*a,p)+*a+(k-*a<0?p:0));
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```

4.2 PeriodicSearch() Template Class

T	**T** must be a sortable data type.

4.3 PeriodicSearch() Parameters

a	**a** points to the beginning of the array that will be searched. The array should be sorted into ascending order.

b	**b** points to one element past the end of the array that will be searched. Thus, **b** is used to define the end of the search region but is not included in the search. **b** should be greater than **a**.
k	**k** is the key on which the search is based. Note that before the function returns, **k** is set to k', where $k' =$ **k** $+ n$**p**.
p	**p** is the period of the function associated with the array. Values for **p** are typically chosen such that **p** > *(**b**-1) - ***a**. In no case should **p** be equal to zero.

4.4 PeriodicSearch() Return Value

The PeriodicSearch() function returns a pointer that satisfies the requirements for **c** in Eq. 4.

4.5 PeriodicSearch() Example

The following example begins by creating a pair of arrays that are used to store independent and dependent variables that approximate Eq. 6.

$$y(x) = \sin(2\pi x/5) + 1. \tag{6}$$

The BinarySearch() and PeriodicSearch() functions are then used to retrieve dependent variables based on a selection of independent variables. Figure 2 presents a pair of graphs that display the results of the searches.

```
#include <cstdio>//...................................freopen(),printf(),stdout
#include "y_interp.h"//..................................yInterp,<cmath>{sin()}
int main(){//<==============COMPARISON BETWEEN BinarySearch() & PeriodicSearch()
  freopen("sine.csv","w",stdout);
  double X[20];/*<-*/for(int i=0;i<20;++i)X[i]=5*i/20.+5;
  double Y[20];/*<-*/for(int i=0;i<20;++i)Y[i]=sin(2*3.14159*X[i]/5)+1;
  printf("x,y - BinarySearch()),y - PeriodicSearch()\n");
  for(double x=-5,k;x<15;x+=.1){
    printf("%f,%f,",x,Y[yInterp::BinarySearch(X+1,X+20,x)-X]);
    printf("%f\n",Y[yInterp::PeriodicSearch(X,X+20,k=x,5.)-X]);}
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```

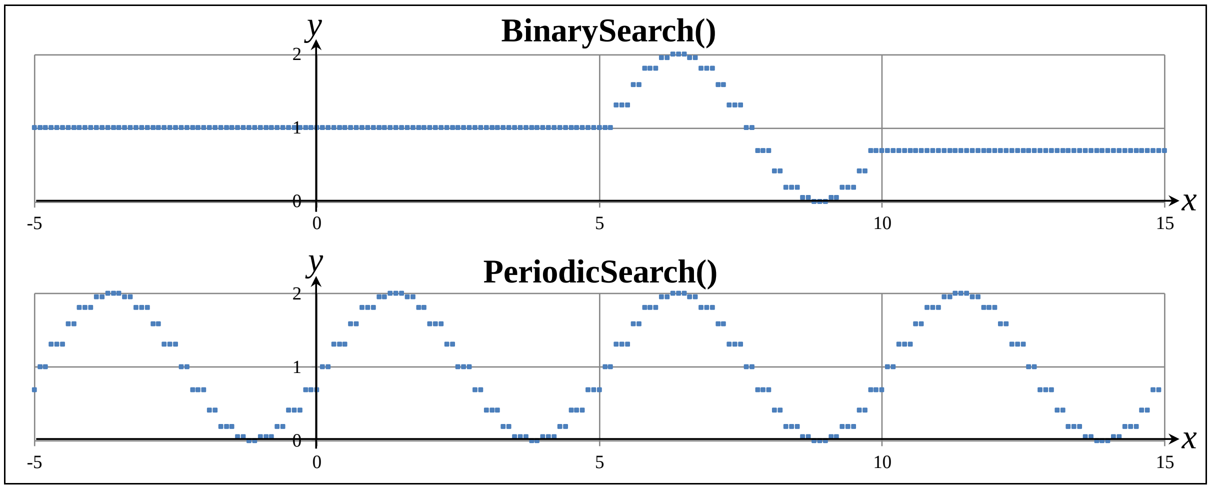


Fig. 2   A comparison of the Search() and PeriodicSearch() functions

5. Performing Nearest-Neighbor Interpolations: The NNInterp() Function

Suppose that some function $y(x)$ is represented by a finite set of points (x_i, y_i), such as is shown in Fig. 3.

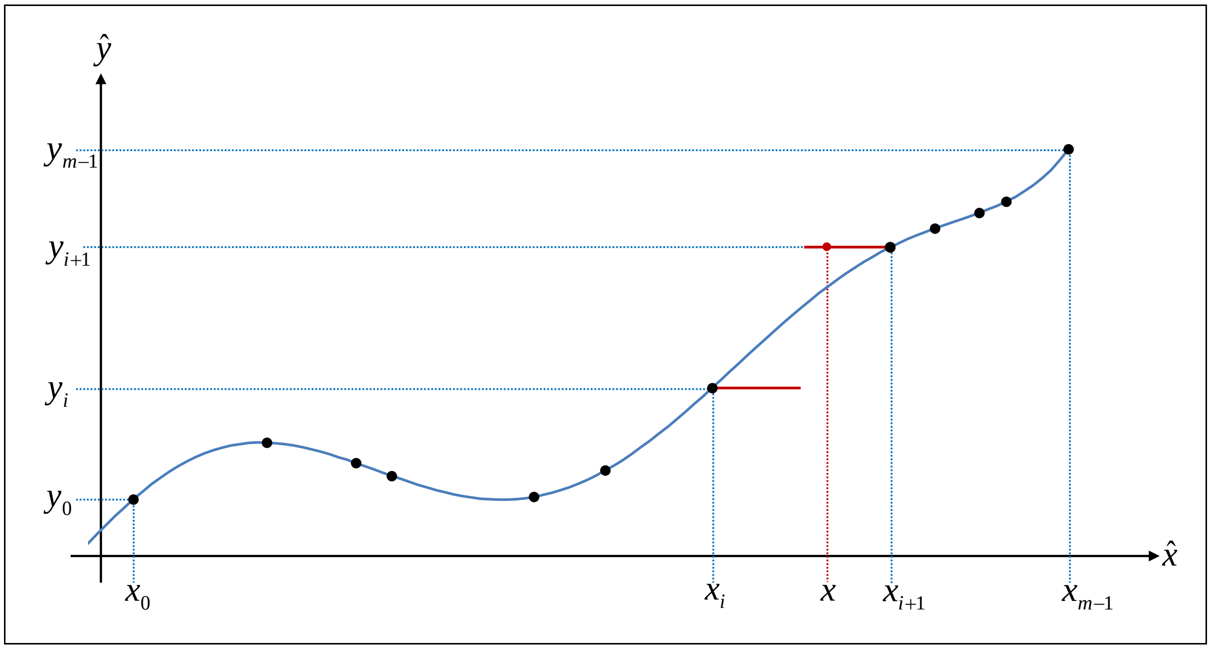


Fig. 3 $y(x)$ represented by a finite number of points, with nearest-neighbor interpolation shown in red

The NNInterp() function uses Eq. 7, represented by the solid red line in Fig. 3, to approximate $y(x)$:

$$y(x) = \begin{cases} y_{i+1} & \text{for } x > \dfrac{x_i + x_{i+1}}{2} \\ y_i & \text{otherwise} \end{cases} . \qquad (7)$$

5.1 NNInterp() Code

```
template<class T>T NNInterp(//<==================NEAREST-NEIGHBOR INTERPOLATOR
    const T*X,//<--------------BRACKETING X VALUES (*X AND X[1] MUST BE VALID)
    const T*Y,//<--------------BRACKETING Y VALUES (*Y AND Y[1] MUST BE VALID)
    T x){//<--------------VALUE TO INTERPOLATE AT (TYPICALLY, *X <= x <= x[1])
  return x>(*X+X[1])/2?Y[1]:*Y;
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```

5.2 NNInterp() Template Class

T	**T** is typically a floating-point data type.

5.3 NNInterp() Parameters

X	**X** points to an array that is used to store values for x_i from Eq. 7. Specifically, ***X** = x_i and **X**[1] = x_{i+1}. Thus, both **X** and **X**+1 must point to valid addresses.
Y	**Y** points to an array that is used to store values for y_i from Eq. 7. Specifically, ***Y** = y_i and **Y**[1] = y_{i+1}. Thus, both **Y** and **Y**+1 must point to valid addresses.
x	**x** represents the independent variable x from Eq. 7.

5.4 NNInterp() Return Value

The NNInterp() function returns y from Eq. 7.

5.5 NNInterp() Simple Example

The following example begins by creating a pair of arrays that are used to store independent and dependent variables that approximate Eq. 6. Next, the BinarySearch() function is used to find a pointer that conforms to Eq. 1, which is then converted to an index. Finally, the NNInterp() function is used to approximate $y(7.180)$.

Note that the value for the BinarySearch() parameter **a** is set to **X**+1 to avoid the possibility of accessing the **X**-1 element. Similarly, the value for the BinarySearch() parameter **b** is set to **X**+19 to avoid the possibility of accessing the **X**+20 element.

```
#include <cstdio>//.................................................printf()
#include "y_interp.h"//.................................yInterp,<cmath>{sin()}
int main(){//<=============A SIMPLE EXAMPLE FOR THE yInterp::NNInterp() FUNCTION
  double X[20];/*<-*/for(int i=0;i<20;++i)X[i]=5*i/20.+5;
  double Y[20];/*<-*/for(int i=0;i<20;++i)Y[i]=sin(2*3.14159*X[i]/5)+1;
  double x=7.18;
  int i=yInterp::BinarySearch(X+1,X+19,x)-X;
  double y=yInterp::NNInterp(X+i,Y+i,x);
  printf("At x=%.3f, y is approximately %.3f.\n",x,y);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```

OUTPUT:

```
At x=7.180, y is approximately 1.309.
```

6. Performing Linear Interpolations: The LinInterp() Function

Suppose that some function $y(x)$ is represented by a finite set of points (x_i, y_i), such as is shown in Fig. 4.

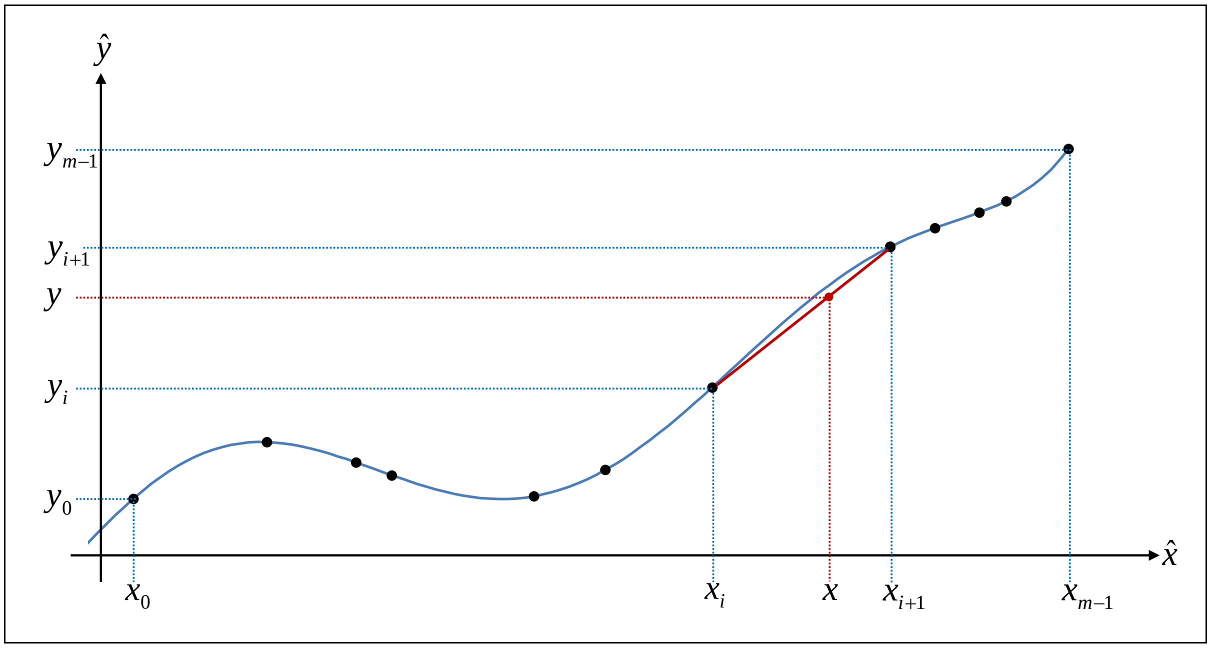


Fig. 4 $y(x)$ represented by a finite number of points, with linear interpolation shown in red

The LinInterp() function uses Eq. 8, represented by the solid red line in Fig. 4, to approximate $y(x)$:

$$y(x) = y_i + (y_{i+1} - y_i)\frac{x - x_i}{x_{i+1} - x_i}. \tag{8}$$

6.1 LinInterp() Code

```
template<class T>T LinInterp(//<==========================LINEAR INTERPOLATOR
    const T*X,//<--------------BRACKETING X VALUES (*X AND X[1] MUST BE VALID)
    const T*Y,//<--------------BRACKETING Y VALUES (*Y AND Y[1] MUST BE VALID)
    T x){//<--------------VALUE TO INTERPOLATE AT (TYPICALLY, *X <= x <= x[1])
  return*Y+(Y[1]-*Y)*(x-*X)/(X[1]-*X);
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```

6.2 LinInterp() Template Class

T **T** is typically a floating-point data type.

6.3 LinInterp() Parameters

X **X** points to an array that is used to store values for x_i from Eq. 8. Specifically, ***X** = x_i and **X**[1] = x_{i+1}. Thus, both **X** and **X**+1 must point to valid addresses.

Y **Y** points to an array that is used to store values for y_i from Eq. 8. Specifically, ***Y** = y_i and **Y**[1] = y_{i+1}. Thus, both **Y** and **Y**+1 must point to valid addresses.

x **x** represents the independent variable x from Eq. 8.

6.4 LinInterp() Return Value

The LinInterp() function returns y from Eq. 8.

6.5 LinInterp() Simple Example

The following example begins by creating a pair of arrays that are used to store independent and dependent variables that approximate Eq. 6. Next, the BinarySearch() function is used to find a pointer that conforms to Eq. 1, which it then converted to an index. Finally, the LinInterp() function is used to approximate $y(7.180)$.

Note that the value for the BinarySearch() parameter **a** is set to **X**+1 to avoid the possibility of accessing the **X**-1 element. Similarly, the value for the BinarySearch() parameter **b** is set to **X**+19 to avoid the possibility of accessing the **X**+20 element.

```
#include <cstdio>//....................................................printf()
#include "y_interp.h"//..................................yInterp,<cmath>{sin()}
int main(){//<============A SIMPLE EXAMPLE FOR THE yInterp::LinInterp() FUNCTION
  double X[20];/*<-*/for(int i=0;i<20;++i)X[i]=5*i/20.+5;
  double Y[20];/*<-*/for(int i=0;i<20;++i)Y[i]=sin(2*3.14159*X[i]/5)+1;
  double x=7.18;
  int i=yInterp::BinarySearch(X+1,X+19,x)-X;
  double y=yInterp::LinInterp(X+i,Y+i,x);
  printf("At x=%.3f, y is approximately %.3f.\n",x,y);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```

OUTPUT:

```
At x=7.180, y is approximately 1.387.
```

7. Performing Cubic Interpolations: The CubeInterp() Function

A cubic Hermite spline is a third-degree-polynomial interpolating function that is uniquely determined by 2 endpoint positions ($\vec{p}_0$ and $\vec{p}_1$) and tangent vectors at the 2 endpoint positions ($\vec{m}_0$ and $\vec{m}_1$, respectively):

$$\vec{p}(t) = H_0(t)\vec{p}_0 + H_1(t)\vec{m}_0 + H_2(t)\vec{p}_1 + H_3(t)\vec{m}_1 . \tag{9}$$

$H_0(t)$, $H_1(t)$, $H_2(t)$, and $H_3(t)$ are known as Hermite basis functions and are given by Eqs. 10–13. t is the linearly scaled distance from $\vec{p}_0$ to $\vec{p}_1$ such that at $\vec{p}_0$, $t = 0$ and at $\vec{p}_1$, $t = 1$.

$$H_0(t) \equiv 2t^3 - 3t^2 + 1 . \tag{10}$$

$$H_1(t) \equiv t^3 - 2t^2 + t . \tag{11}$$

$$H_2(t) \equiv -2t^3 + 3t^2 . \tag{12}$$

$$H_3(t) \equiv t^3 - t^2 . \tag{13}$$

Suppose that some function $y(x)$ is represented by a finite set of points (x_i, y_i), such as is shown in Fig. 5.

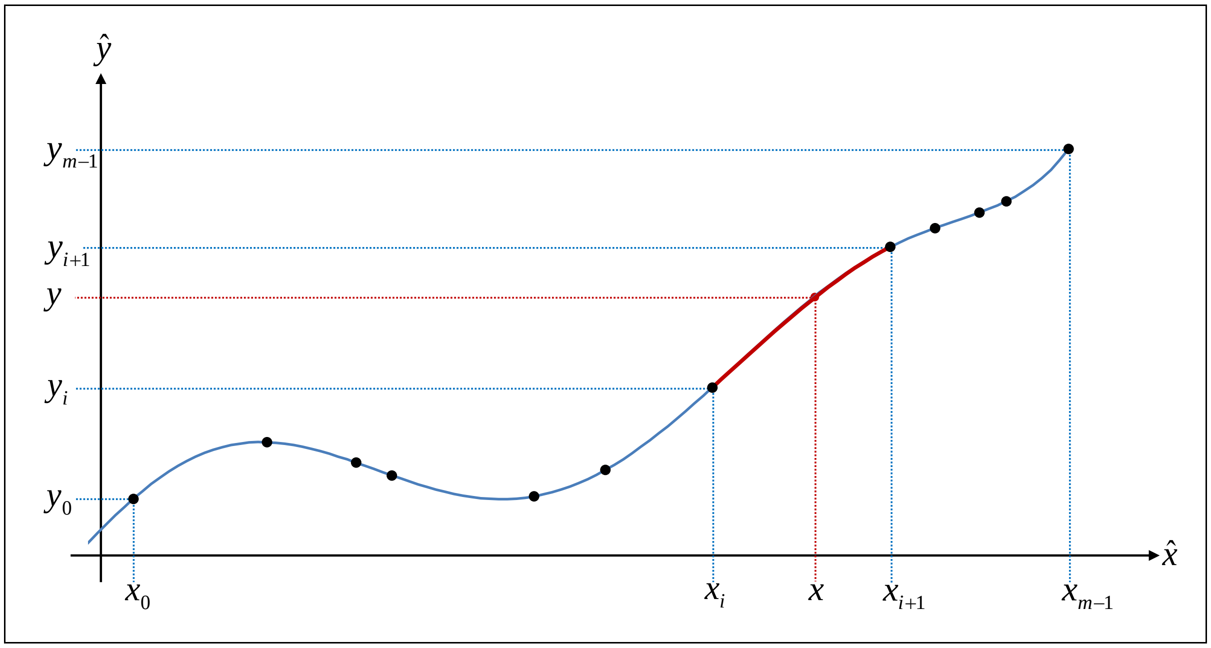


Fig. 5 $y(x)$ represented by a finite number of points, with cubic interpolation shown in red

From Eq. 9 (and noting that $H_0(t) = 1 - H_2(t)$),

$$y(t) = H_1(t)\left.\frac{dy}{dt}\right|_{t=0} + H_2(t)(y_{i+1} - y_i) + H_3(t)\left.\frac{dy}{dt}\right|_{t=1} + y_i \,, \tag{14}$$

Eq. 15 can be used to convert from the scaled independent variable t to the general independent variable x.

$$t = \frac{x - x_i}{x_{i+1} - x_i} \,. \tag{15}$$

Eq. 16, along with the chain rule, can be used to convert the derivatives.

$$\frac{dt}{dx} = \frac{1}{x_{i+1} - x_i} \,. \tag{16}$$

Thus,

$$\left.\frac{dy}{dt}\right|_{t=0} = (x_{i+1} - x_i)m_i \,, \tag{17}$$

and

$$\left.\frac{dy}{dt}\right|_{t=1} = (x_{i+1} - x_i)m_{i+1} \,, \tag{18}$$

where

$$m_i \equiv \left.\frac{dy}{dx}\right|_{x=x_i}. \tag{19}$$

Substituting Eqs. 11–13, 17, and 18 into Eq. 14, then regrouping terms, results in the form of the cubic interpolating function that is used by the CubeInterp() function:

$$y = y_i + (x - x_i)[\{(m_i + m_{i+1})t - (2m_i + m_{i+1})\}t + m_i] + (y_{i+1} - y_i)(3 - 2t)t^2. \tag{20}$$

7.1 CubeInterp() Code

```
template<class T>T CubeInterp(//<==========CUBIC (HERMITE SPLINE) INTERPOLATOR
    const T*X,//<--------------BRACKETING X VALUES (*X AND X[1] MUST BE VALID)
    const T*Y,//<--------------BRACKETING Y VALUES (*Y AND Y[1] MUST BE VALID)
    T x,//<---------------VALUE TO INTERPOLATE AT (TYPICALLY, *X <= x <= x[1])
    T m0,T m1){//<---------------------------------------SLOPES AT *X AND X[1]
  T t=(x-*X)/(X[1]-*X);
  return*Y+(x-*X)*(((m0+m1)*t-(2*m0+m1))*t+m0)+(Y[1]-*Y)*(3-2*t)*t*t;
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```

7.2 CubeInterp() Template Class

T **T** is typically a floating-point data type.

7.3 CubeInterp() Parameters

X **X** points to an array that is used to store values for x_i from Eq. 20. Specifically, ***X** = x_i and **X**[1] = x_{i+1}. Thus, both **X** and **X**+1 must point to valid addresses.

Y **Y** points to an array that is used to store values for y_i from Eq. 20. Specifically, ***Y** = y_i and **Y**[1] = y_{i+1}. Thus, both **Y** and **Y**+1 must point to valid addresses.

m0 **m0** represents m_i, the slope of the interpolating function at x_i.

m1 **m1** represents m_{i+1}, the slope of the interpolating function at x_{i+1}.

x **x** represents the independent variable x from Eq. 20.

7.4 CubeInterp() Return Value

The CubeInterp() function returns y from Eq. 20.

7.5 CubeInterp() Simple Example

The following example begins by creating a pair of arrays that are used to store independent and dependent variables that approximate Eq. 6. Next, the BinarySearch() function is used to find a pointer that conforms to Eq. 1, which is then converted to an index. Finally, the CubeInterp()

function is used to approximate $y(7.180)$. For simplicity, m_i and m_{i+1} have been set to zero. Section 8 presents a function that can be used to find values for m_i and m_{i+1}.

Note that the value for the BinarySearch() parameter **a** is set to **X**+1 to avoid the possibility of accessing the **X**-1 element. Similarly, the value for the BinarySearch() parameter **b** is set to **X**+19 to avoid the possibility of accessing the **X**+20 element.

```
#include <cstdio>//.................................................printf()
#include "y_interp.h"//................................yInterp,<cmath>{sin()}
int main(){//<===========A SIMPLE EXAMPLE FOR THE yInterp::CubeInterp() FUNCTION
  double X[20];/*<-*/for(int i=0;i<20;++i)X[i]=5*i/20.+5;
  double Y[20];/*<-*/for(int i=0;i<20;++i)Y[i]=sin(2*3.14159*X[i]/5)+1;
  double x=7.18;
  int i=yInterp::BinarySearch(X+1,X+19,x)-X;
  double y=yInterp::CubeInterp(X+i,Y+i,x,0.,0.);
  printf("At x=%.3f, y is approximately %.3f.\n",x,y);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```

OUTPUT:

```
At x=7.180, y is approximately 1.362.
```

8. Cardinal Splines: The CardinalSlope() Function

The CardinalSlope() function uses Eq. 21 to calculate the CubeInterp()-function input parameters **m0** and **m1**.

$$m_i = (1-t)\frac{y_{i+1} - y_{i-1}}{x_{i+1} - x_{i-1}}. \tag{21}$$

t is sometimes referred to as a tension parameter and is typically limited to the interval [0,1].

8.1 CardinalSlope() Code

```
  template<class T>T CardinalSlope(//<========CALCULATES SLOPES FOR CubeInterp()
      const T*X,//<-----------BRACKETING X VALUES (X[-1] AND X[1] MUST BE VALID)
      const T*Y,//<-----------BRACKETING Y VALUES (Y[-1] AND Y[1] MUST BE VALID)
      T t){//<--------------------------------------------------TENSION PARAMETER
    return(1-t)*(Y[1]-Y[-1])/(X[1]-X[-1]);
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```

8.2 CardinalSlope() Template Class

T	**T** is typically a floating-point data type.

8.3 CardinalSlope() Parameters

X	**X** points to an array that is used to store values for x_i from Eq. 21. Specifically, **X**[-1] = x_{i-1} and **X**[1] = x_{i+1}. Thus, both **X**-1 and **X**+1 must point to valid addresses.
Y	**Y** points to an array that is used to store values for y_i from Eq. 21. Specifically, **Y**[-1] = y_{i-1} and **Y**[1] = y_{i+1}. Thus, both **Y**-1 and **Y**+1 must point to valid addresses.
t	**t** represents t, the tension parameter from Eq. 21.

Because the CardinalSlope() function looks at points to either side of the bracketing points, extra care needs to be taken when selecting **X** and **Y**.

8.4 CardinalSlope() Return Value

The CardinalSlope() function returns m_i from Eq. 21.

8.5 CardinalSlope() Simple Example

The following example begins by creating a pair of arrays that are used to store independent and dependent variables that approximate Eq. 6. Next, the BinarySearch() function is used to find a pointer that conforms to Eq. 1, which is then converted to an index. Finally, the CubeInterp() function is used to approximate $y(7.180)$.

Note that the value for the BinarySearch() parameter **a** has been set to **X**+1 to avoid the possibility of accessing the **X**-1 element. Similarly, the value for the BinarySearch() parameter **b** has been set to **X**+19 to avoid the possibility of accessing the **X**+20 element.

```
#include <cstdio>//.....................................................printf()
#include "y_interp.h"//...................................yInterp,<cmath>{sin()}
int main(){//<========A SIMPLE EXAMPLE FOR THE yInterp::CardinalSlope() FUNCTION
  double X[20];/*<-*/for(int i=0;i<20;++i)X[i]=5*i/20.+5;
  double Y[20];/*<-*/for(int i=0;i<20;++i)Y[i]=sin(2*3.14159*X[i]/5)+1;
  double x=7.18;
  int i=yInterp::BinarySearch(X+1,X+19,x)-X;
  double m0=i==0||i>17?0:yInterp::CardinalSlope(X+i,Y+i,0.),
    m1=i==0||i>17?0:yInterp::CardinalSlope(X+i+1,Y+i+1,0.);
  double y=yInterp::CubeInterp(X+i,Y+i,x,m0,m1);
  printf("At x=%.3f, y is approximately %.3f.\n",x,y);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```

OUTPUT:

```
At x=7.180, y is approximately 1.391.
```

9. Example: Comparing Interpolating Functions

The following example code creates a comma-separated text file that can be used to create the graph shown in Fig. 6. The plot relating to the CubeInterp() function relies on slopes that are calculated using the CardinalSlope() function (except for the endpoints, where the slopes are set to zero).

```
#include <cstdio>//..................................freopen(),printf(),stdout
#include "y_interp.h"//...................................................yInterp
int main(){//<================COMPARISON BETWEEN VARIOUS INTERPOLATION FUNCTIONS
  freopen("interp.csv","w",stdout);//.................redirect output to a file
  double X[8]={2,7,13,19,22,23,28,37},Y[8]={2,6,6,2,9,5,4,7};
  double m[8]={0};/*<-*/for(int i=1;i<7;++i)
    m[i]=yInterp::CardinalSlope(X+i,Y+i,0.);
  printf("#x,y,x,y1,y2,y3\n");
  for(int k=0,n=20000;k<n;++k){
    double x=50/(n-1.)*k;
    int i=yInterp::BinarySearch(X+1,X+8-1,x)-X;//..................note 0>=i<=6
    double y1=yInterp::NNInterp(X+i,Y+i,x);
    double y2=yInterp::LinInterp(X+i,Y+i,x);
    double y3=yInterp::CubeInterp(X+i,Y+i,x,m[i],m[i+1]);
    k<8?printf("%f,%f,",X[k],Y[k]):printf("-,-,");
    printf("%f,%f,%f,%f\n",x,y1,y2,y3);}
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```

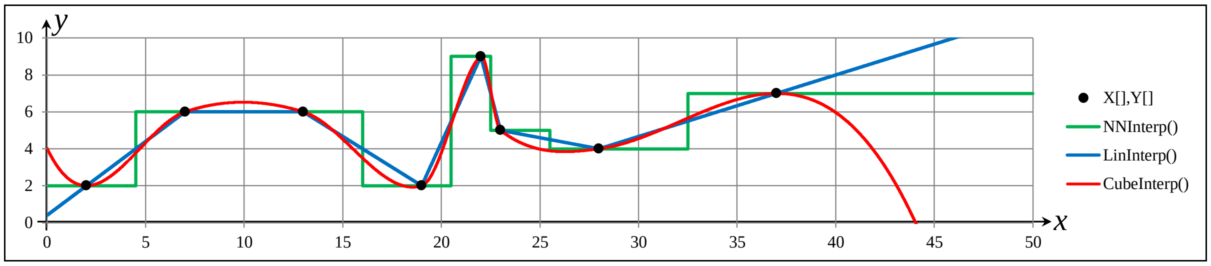


Fig. 6 Comparison of the NNInterp(), LinInterp(), and CubeInterp() functions

10. Example: Performing Periodic Interpolations

The following example code begins by setting up a set of points that are used to represent a periodic function with a period equal to 10.0. Next, the NNInterp(), LinInterp(), and CubInterp() functions are used to create values for a comma-separated text file. The graph shown in Fig. 7 represents the contents of the output file.

```
#include <cstdio>//...................................freopen(),printf(),stdout
#include "y_interp.h"//...................................................yInterp
int main(){//<=====COMPARISON BETWEEN VARIOUS INTERPOLATION FUNCTIONS (PERIODIC)
  freopen("interp_periodic.csv","w",stdout);//.........redirect output to a file
  double X[4]={10,14,16,18},Y[4]={6,2,9,5};
  double XP[7]={X[3]-10,X[0],X[1],X[2],X[3],X[0]+10,X[1]+10};
  double YP[7]={Y[3],Y[0],Y[1],Y[2],Y[3],Y[0],Y[1]};
  double m[5];/*<-*/for(int i=0;i<5;++i)
    m[i]=yInterp::CardinalSlope(XP+1+i,YP+1+i,0.);
  printf("#x,y,x,y1,y2,y3\n");
  int j=0;
  for(double x=-10,k;x<70;x+=.1,++j){
    int i=yInterp::PeriodicSearch(X,X+4,k=x,10.)-X;
    double y1=yInterp::NNInterp(XP+1+i,YP+1+i,k);
    double y2=yInterp::LinInterp(XP+1+i,YP+1+i,k);
    double y3=yInterp::CubeInterp(XP+1+i,YP+1+i,k,m[i],m[i+1]);
    j<4?printf("%f,%f,",X[j],Y[j]):printf("-,-,");
    printf("%f,%f,%f,%f\n",x,y1,y2,y3);}
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```

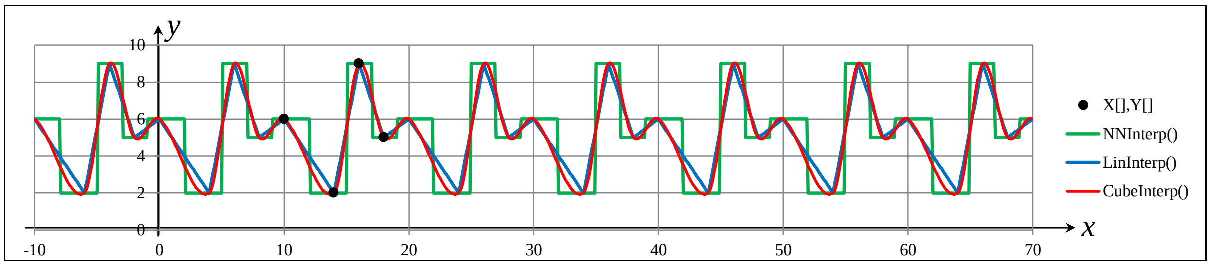


Fig. 7   NNInterp(), LinInterp(), and CubeInterp() functions used to create periodic plots

11. Example: Determining Interpolation Performance

The following example can be used to test the performance of the NNInterp(), LinInterp(), and CubeInterp() functions.

The output was generated by compiling the code using Microsoft's Visual Studio C++ 2010 Express compiler, with the output set to “release” mode. Surprisingly, for this scenario, the LinInterp() and CubeInterp() functions outperform the NNInterp() function.

```
#include <cstdio>//..................................................printf()
#include <ctime>//.......................................clock(),CLOCKS_PER_SEC
#include "y_interp.h"//................................................yInterp
#include "y_random.h"//................................................yRandom
int main(){//<====PERFORMANCE COMPARISON BETWEEN VARIOUS INTERPOLATION FUNCTIONS
  const int N=100000000;
  double X[2]={0,1},Y[2]={0,10};
  unsigned I[625];/*<-*/yRandom::Initialize(I,1);//....state of Mersenne twister
  double R[N];/*<-*/for(int i=0;i<N;++i)R[i]=yRandom::RandU(I,0,1);
  double S=0,t=clock();
  for(int i=0;i<N;++i)S+=yInterp::NNInterp(X,Y,R[i]);
  printf("Using NNInterp():\n  y_avg=%.6f,t_avg=%.2f ns\n\n",S/N,
    (clock()-t)/CLOCKS_PER_SEC/N*1E9),t=clock(),S=0;
  yRandom::Initialize(I,1);
  for(int i=0;i<N;++i)S+=yInterp::LinInterp(X,Y,R[i]);
  printf("Using LinInterp():\n  y_avg=%.6f,t_avg=%.2f ns\n\n",S/N,
    (clock()-t)/CLOCKS_PER_SEC/N*1E9),t=clock(),S=0;
  yRandom::Initialize(I,1);
  for(int i=0;i<N;++i)S+=yInterp::CubeInterp(X,Y,R[i],10.,10.);
  printf("Using CubeInterp():\n  y_avg=%.6f,t_avg=%.2f ns\n\n",S/N,
    (clock()-t)/CLOCKS_PER_SEC/N*1E9);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```

OUTPUT:

```
Using NNInterp():
  y_avg=4.999344,t_avg=4.52 ns

Using LinInterp():
  y_avg=4.999774,t_avg=2.50 ns

Using CubeInterp():
  y_avg=4.999709,t_avg=3.43 ns
```

12. Example: Interpolating in Two Dimensions

The NNInterp(), LinInterp(), and CubeInterp() functions can be used to perform interpolations in 2 (and higher) dimensions.

Suppose that $z_{i,j} \equiv z(x_i, y_j)$. To interpolate in 2 dimensions, begin by performing 2 interpolations in the $\hat{y}$ direction, 1 at x_i and 1 at x_{i+1} (shown in green in Fig. 8). Define z_a to be the interpolation at x_i and z_b to be the interpolation at x_{i+1}. Complete the interpolation by interpolating between z_a and z_b in the $\hat{x}$ direction (shown in red in Fig. 8).

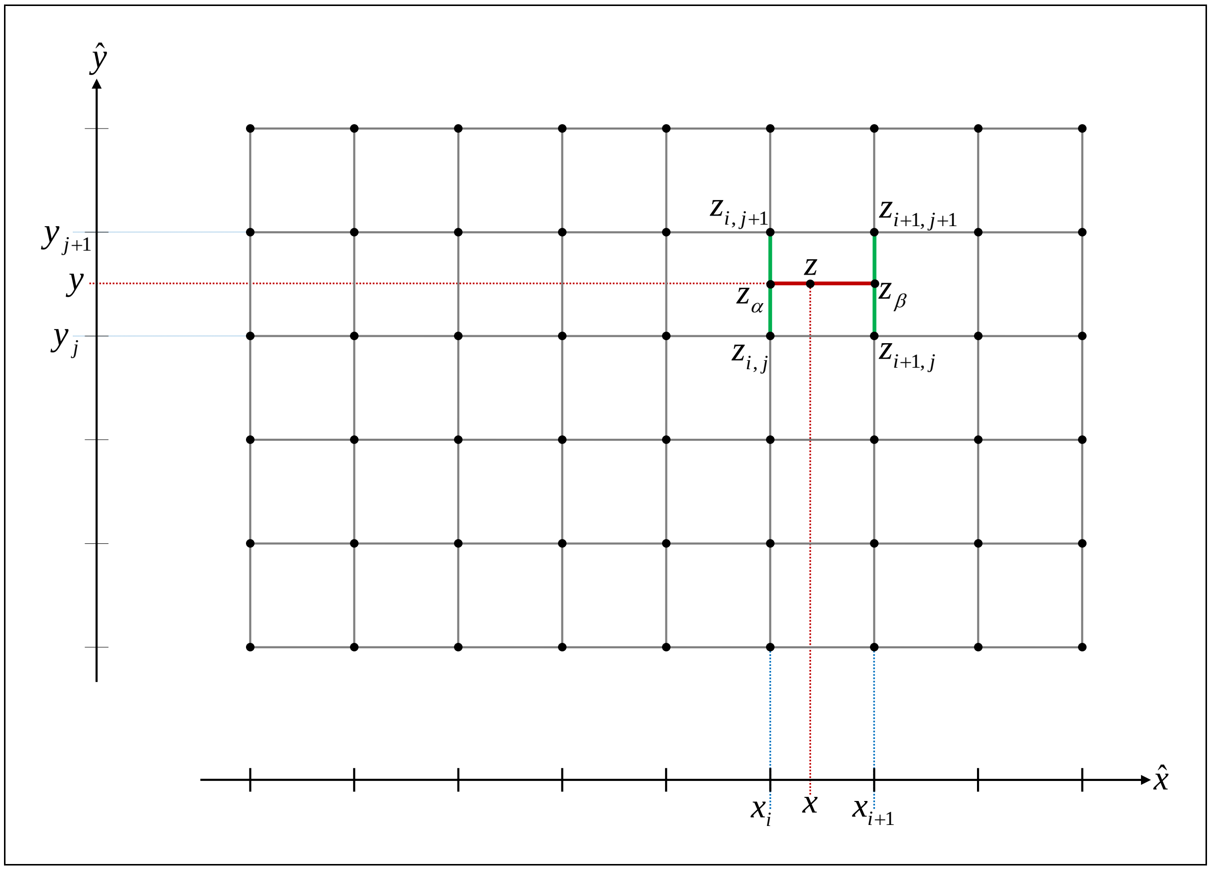


Fig. 8 Interpolating in 2 dimensions

The following example code uses the yBmp namespace to create 3 images (presented in Fig. 9) that show the results of using the NNInterp(), LinInterp(), and CubeInterp() functions to interpolate in 2 dimensions.

```
#include "y_interp.h"//..................................yInterp,<cmath>{fabs()}
#include "y_bmp.h"//........................................................yBmp
inline void Rainbow(//<========================================RAINBOW COLOR MAP
    unsigned char C[3],//<-----------------------------OUTPUT COLOR (CALCULATED)
    double x,//<----------------------VALUE FOR WHICH A COLOR WILL BE CALCULATED
    double min,double max){//<---------------MINIMUM AND MAXIMUM SCALED VALUES
  if(x<min){C[0]=C[1]=C[2]=0;/*&*/return;}//.......set too small values to black
  if(x>max){C[0]=C[1]=C[2]=255;/*&*/return;}//.....set too large values to white
  x=(1-(x-min)/(max-min))*8;//......................remap x to a range of 8 to 0
  C[0]=int((3<x&&x<5||x>7     ?-fabs(x/2-3)+1.5:5<=x&&x<=7?1:0)*255);//.....blue
  C[1]=int((1<x&&x<3||5<x&&x<7?-fabs(x/2-2)+1.5:3<=x&&x<=5?1:0)*255);//....green
  C[2]=int((     x<1||3<x&&x<5?-fabs(x/2-1)+1.5:1<=x&&x<=3?1:0)*255);//......red
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~15JUL2014~~~~~~
int main(){//<======CREATE AN IMAGE FROM A SURFACE USING INTERPOLATING FUNCTIONS
  const int m=4,n=4,r=250;//.........# of x values, # of y values, pixel scaling
  double X[4]={0,1,2,3},Y[4]={0,1,2,3};
  double Z[m][n]={1,1,1,0 , 1,0,0,1 , 0,1,0,1 , 0,0,0,0};
  unsigned char*I=yBmp::NewImage(m*r,n*r,255);
  for(int q=0;q<m*r;++q)for(int p=0;p<n*r;++p){
```

```
    double x=q*(m-1)*1./(m*r-1),y=p*(n-1)*1./(n*r-1);
    int i=yInterp::BinarySearch(X+1,X+m-1,x)-X,
      j=yInterp::BinarySearch(Y+1,Y+n-1,y)-Y;
    double ZI[2]={yInterp::NNInterp(Y+j,Z[i]+j,y),
      yInterp::NNInterp(Y+j,Z[i+1]+j,y)};
    double z=yInterp::NNInterp(X+i,ZI,x);
    Rainbow(yBmp::GetPixel(I,q,p),z,-.2,1.2);}
  yBmp::WriteBmpFile("nearest_neighbor.bmp",I);
  for(int q=0;q<m*r;++q)for(int p=0;p<n*r;++p){
    double x=q*(m-1)*1./(m*r-1),y=p*(n-1)*1./(n*r-1);
    int i=yInterp::BinarySearch(X+1,X+m-1,x)-X,
      j=yInterp::BinarySearch(Y+1,Y+n-1,y)-Y;
    double ZI[2]={yInterp::LinInterp(Y+j,Z[i]+j,y),
      yInterp::LinInterp(Y+j,Z[i+1]+j,y)};
    double z=yInterp::LinInterp(X+i,ZI,x);
    Rainbow(yBmp::GetPixel(I,q,p),z,-.2,1.2);}
  yBmp::WriteBmpFile("linear.bmp",I);
  for(int q=0;q<m*r;++q)for(int p=0;p<n*r;++p){
    double x=q*(m-1)*1./(m*r-1),y=p*(n-1)*1./(n*r-1);
    int i=yInterp::BinarySearch(X+1,X+m-1,x)-X,
      j=yInterp::BinarySearch(Y+1,Y+n-1,y)-Y;
    double ZI[2]={yInterp::CubeInterp(Y+j,Z[i]+j,y,0.,0.),
      yInterp::CubeInterp(Y+j,Z[i+1]+j,y,0.,0.)};
    double z=yInterp::CubeInterp(X+i,ZI,x,0.,0.);
    Rainbow(yBmp::GetPixel(I,q,p),z,-.2,1.2);}
  yBmp::WriteBmpFile("cubic.bmp",I);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```

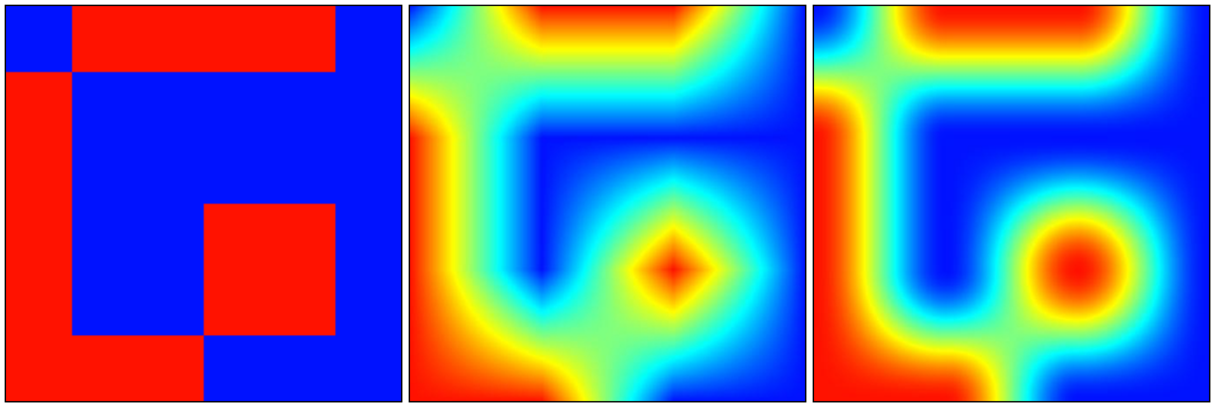


Fig. 9 2-dimensional interpolations using NNInterp() (left), LinInterp() (center), and CubeInterp() (right)

13. Polynomial Curve Fitting: The PolyFit() Function

Suppose that some set of n ordered pairs (x_k, y_k) represents a set of measurements, with x_k being a value for an independent variable and y_k being a value for a dependent variable.

Furthermore, suppose that Eq. 22 represents a best-fit equation for the ordered pairs, where the coefficients (c_i) are unknown.

$$y(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_i x^i + \cdots + c_d x^d \,. \tag{22}$$

Eq. 23 can be used to find the coefficients for Eq. 22:[6]

$$\begin{bmatrix} n & \sum_{k=0}^{k<n} x_k & \cdots & \sum_{k=0}^{k<n} x_k^d \\ \sum_{k=0}^{k<n} x_k & \sum_{k=0}^{k<n} x_k^2 & \cdots & \sum_{k=0}^{k<n} x_k^{d+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{k=0}^{k<n} x_k^d & \sum_{k=0}^{k<n} x_k^{d+1} & \cdots & \sum_{k=0}^{k<n} x_k^{2d} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_d \end{bmatrix} = \begin{bmatrix} \sum_{k=0}^{k<n} y_k \\ \sum_{k=0}^{k<n} x_k y_k \\ \vdots \\ \sum_{k=0}^{k<n} x_k^d y_k \end{bmatrix} \tag{23}$$

The PolyFit() function uses Gaussian elimination with backward substitution to solve Eq. 23.

13.1 PolyFit() Code

```
template<class T>void PolyFit(//<=========FITS A POLYNOMIAL TO A SET OF POINTS
    const T*X,//<-------INDEPENDENT-VARIABLE VALUES (EACH X[i] MUST BE UNIQUE)
    const T*Y,//<-------------------DEPENDENT-VARIABLE VALUES (SAME SIZE AS X)
    int n,//<-------------------------------------NUMBER OF ELEMENTS IN X OR Y
    int m,//<----------------NUMBER OF COEFFICIENTS IN THE BEST-FIT POLYNOMIAL
    T*C){//<------STORAGE FOR COEFFICIENTS (SIZE=m) y=C[0]+C[1]*x+C[2]*x^2+...
  T**A=new T*[m];/*<-*/for(int i=0,j,k;i<m;++i){//...........augmented matrix
    for(A[i]=new T[m+1],j=0;j<m+1;++j)A[i][j]=0;
    for(k=0;k<n;++k)for(A[i][m]+=pow(X[k],i)*Y[k],j=0;j<m;++j)
      A[i][j]+=pow(X[k],i+j);}
  for(int i=1,j,k;i<m;++i)for(k=0;k<i;++k)for(j=m;j>=0;--j)//........Gaussian
     A[i][j]-=A[i][k]*A[k][j]/A[k][k];//                          elimination
  for(int i=m-1,j;i>=0;--i)for(C[i]=A[i][m]/A[i][i],j=i+1;j<m;++j)//..backward
    C[i]-=C[j]*A[i][j]/A[i][i];//                                substitution
  for(int i=0;i<m;++i)delete[]A[i];/*&*/delete[]A;
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```

13.2 PolyFit() Template Classes

T	**T** is typically a floating-point data type.

13.3 PolyFit() Parameters

X	**X** points to an array that is used to store the **n** x_k values that are specified in Eq. 23, where **X**={ $x_0, x_1, \cdots, x_k, \cdots, x_{n-1}$ }. Each x_k value must be unique.
Y	**Y** points to an array that is used to store the **n** y_k values that are specified in Eq. 23, where **Y**={ $y_0, y_1, \cdots, y_k, \cdots, y_{n-1}$ }.
n	**n** specifies n, the number of x_k (or y_k) values.

m **m** specifies the number of coefficients in the array that is pointed to by **C**. Thus, **m** = $d+1$, where d is the degree of the fitting polynomial (see Eq. 22).

C **C** points to storage for an array that is used to store the c_i coefficients (**C**={ $c_0, c_1, \cdots, c_i, \cdots, c_d$ }). **C** must point to an array with storage for at least **m** elements. The elements pointed to by **C** are calculated by the PolyFit() function.

Note that if **m** is too large, or if the values pointed to by **X** are too close together, then the PolyFit() function may return coefficients that do not accurately describe a best-fit curve. It is always best to plot the best-fit curve against measured data to verify the quality of the fit. It is also a good idea to calculate the coefficient of determination (R^2), which is a measurement of the quality of the fit.

$$R^2 \equiv 1 - \frac{\sum_{k=0}^{k<n} (y_k - f(x_k))^2}{\sum_{k=0}^{k<n} (y_k - \bar{y})^2}, \qquad (24)$$

where

$$\bar{y} \equiv \frac{1}{n} \sum_{k=0}^{k<n} y_i . \qquad (25)$$

R^2 values are typically in the interval [0,1], with 1 indicating a perfect fit.

13.4 PolyFit() Simple Example

The following example first defines a polynomial, then uses that polynomial to calculate a set of (x_k, y_k) ordered pairs. The PolyFit() function is used to calculate the coefficients for a polynomial that best fits the set of ordered pairs. The calculated coefficients are shown to be identical (at least to 6 decimal places) to the coefficients of the original polynomial.

```
#include <cstdio>//.....................................................printf()
#include "y_interp.h"//.................................................yInterp
int main(){//<==============A SIMPLE EXAMPLE FOR THE yInterp::PolyFit() FUNCTION
  double a=-308,b=177,c=-33,d=2;
  double X[10],Y[10];
  for(int i=0;i<10;++i){
    X[i]=i,Y[i]=a+b*i+c*i*i+d*i*i*i;
    printf("%f,%f\n",X[i],Y[i]);}
  double C[4];/*<-*/yInterp::PolyFit(X,Y,10,4,C);
  printf("\n");
  for(int i=0;i<4;++i)printf("%f\n",C[i]);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```

OUTPUT:

```
0.000000,-308.000000
1.000000,-162.000000
2.000000,-70.000000
3.000000,-20.000000
4.000000,0.000000
5.000000,2.000000
6.000000,-2.000000
7.000000,0.000000
8.000000,20.000000
9.000000,70.000000

-308.000000
177.000000
-33.000000
2.000000
```

14. Example: Exponential Fits Using the PolyFit() Function

The PolyFit() function can be used to fit nonpolynomial equations to measured data. Suppose Eq. 26 is suspected to be a good fit for some set of data points $\left(x_k, y_k\right)$.

$$y = Ae^{Bx} . \tag{26}$$

The PolyFit() function can be used to find the parameters A and B.

Begin by defining $y' = \ln(y)$. Next, use the PolyFit() function to find c_0 and c_1 (the parameters for a straight line) for the data set $\left(x_k, y'_k\right)$. Thus,

$$y' = \ln(y) = c_0 + c_1 x . \tag{27}$$

Solving for y,

$$y = e^{c_0 + c_1 x} = e^{c_0} e^{c_1 t} . \tag{28}$$

Thus,

$$A = e^{c_0} \text{ and } B = c_1 . \tag{29}$$

The following example code uses the PolyFit() function to fit Eq. 26 to beginning-of-year values for the Dow Jones Industrial Average for the decade 1986–1995. Values were obtained from the file "^dji_d.csv," which was downloaded from http://stooq.com.[7] The yIo2 namespace was used to read and parse the data file. The results are presented in Fig. 10.

```
#include <cstdio>//...................................freopen(),printf(),stdout
#include <vector>//..........................................................vector
#include "y_interp.h"//.............................yInterp,<cmath>{exp(),pow()}
#include "y_io_2.h"//.........................................................yIo
int main(){//<======FITTING AN EXPONENTIAL EQUATION USING THE PolyFit() FUNCTION
  //--------------------------READ AND PARSE FILE-------------------------------
  char*s=yIo2::ReadTextFile("^dji_d.csv");
  std::vector<std::vector<char*> >S=yIo2::Parse2D(s,"/ \t-,");
  //-----------------------INTERPRET DATES AND VALUES--------------------------
  std::vector<double>X,Y;
  for(int i=2,y;(y=atoi(S[i][0]))<1996;++i)
    if(y!=atoi(S[i-1][0])&&y>1984)X.push_back(y),Y.push_back(atof(S[i][6]));
  delete[]s;
  //-------------------------PERFORM EXPONENTIAL FIT----------------------------
  std::vector<double>Yp(X.size());
  for(int i=0,m=X.size();i<m;++i)Yp[i]=log(Y[i]);
  double*a=&X[0],*b=&Yp[0];
  double C[2];/*<-*/yInterp::PolyFit(a,b,X.size(),2,C);
  double y_bar=0;/*<-*/for(int k=0;k<10;++k)y_bar+=Y[k];/*&*/y_bar/=10;
  double SS_res=0,SS_tot=0;/*<-*/for(int k=0;k<10;++k)
    SS_res+=pow(Y[k]-exp(C[0]+C[1]*X[k]),2.),SS_tot+=pow(Y[k]-y_bar,2.);
  printf("A=%E\nB=%f\nR^2=%f\n",exp(C[0]),C[1],1-SS_res/SS_tot);
  freopen("exponential_fit.csv","w",stdout);
  for(int i=1,n=X.size();i<n;++i)
    printf("%f,%f,%f\n",X[i],Y[i],exp(C[0]+C[1]*X[i]));
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```

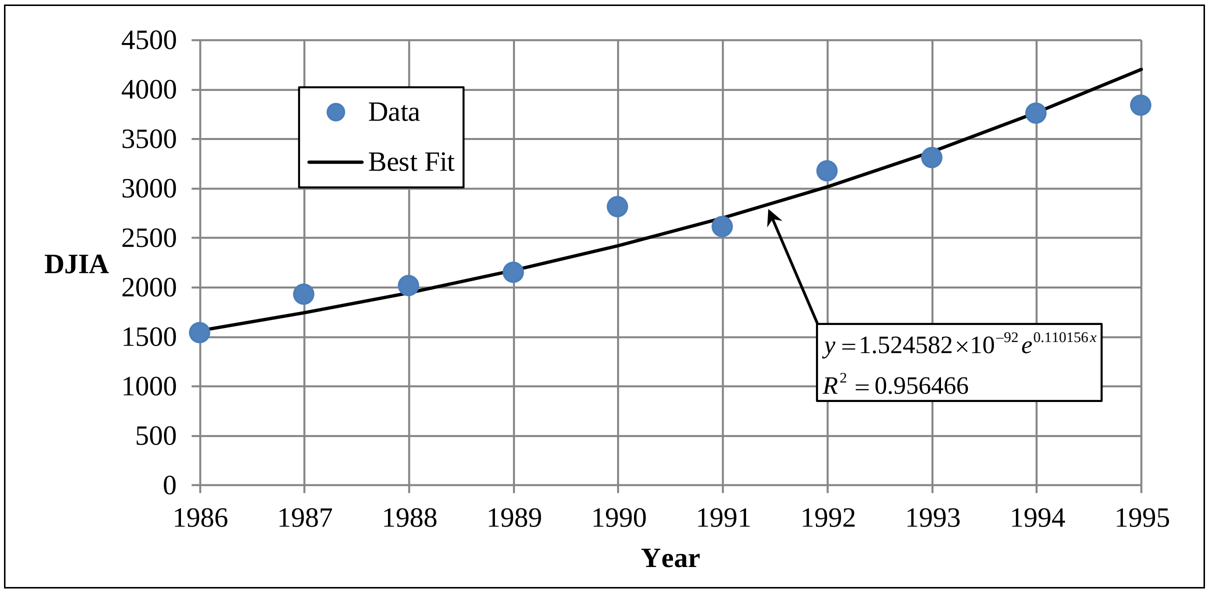


Fig. 10 Best-fit line calculated by the PolyFit() function

15. Code Summary

The following summary sheet presents the yInterp namespace, which contains the BinarySearch(), PeriodicSearch(), NNInterp(), LinInterp(), CubeInterp(), CardinalSlope(), and PolyFit() functions.

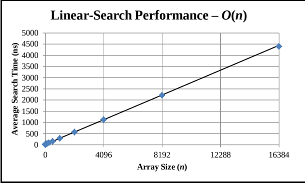
yInterp Summary

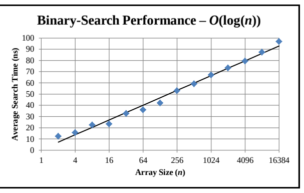
y_interp.h

```
#ifndef Y_INTERP_GUARD//    See Yager, R.J. "Basic Searching, Interpolating, and
#define Y_INTERP_GUARD//           Curve Fitting Algorithms in C++" (ARL-TN-XXX)
#include <cmath>//..................................................fmod(),pow()
namespace yInterp{//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  template<class T>T*BinarySearch(//<======FIND THE POINTER c | *c <= k < *(c+1)
      T*a,T*b,//<--ARRAY START & END POINTS (ARRAY MUST BE SORTED IN INC. ORDER)
      T k){//<-----------------------------------------------------------------KEY
    if(k<*a)return a-1;//.........note that a-1 may point to an invalid address
    for(T*c;k<*--b;k>*c?a=c:b=c+1)c=a+(b-a)/2;/*->*/return b;
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
  template<class T>T*PeriodicSearch(//<=======FOR ARRAYS WITH PERIODIC VARIABLES
      T*a,T*b,//<--STARTING & ENDING POINTS (ARRAY MUST BE SORTED IN INC. ORDER)
      T&k,//<---------------------------------------KEY (WILL BE SET TO k'=k+np)
      T p){//<-----------------------------------------------------PERIOD (p>0)
    return BinarySearch(a,b,k=fmod(k-*a,p)+*a+(k-*a<0?p:0));
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
  template<class T>T NNInterp(//<==================NEAREST-NEIGHBOR INTERPOLATOR
      const T*X,//<--------------BRACKETING X VALUES (*X AND X[1] MUST BE VALID)
      const T*Y,//<--------------BRACKETING Y VALUES (*Y AND Y[1] MUST BE VALID)
      T x){//<--------------VALUE TO INTERPOLATE AT (TYPICALLY, *X <= x <= X[1])
    return x>(*X+X[1])/2?Y[1]:*Y;
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
  template<class T>T LinInterp(//<===========================LINEAR INTERPOLATOR
      const T*X,//<--------------BRACKETING X VALUES (*X AND X[1] MUST BE VALID)
      const T*Y,//<--------------BRACKETING Y VALUES (*Y AND Y[1] MUST BE VALID)
      T x){//<--------------VALUE TO INTERPOLATE AT (TYPICALLY, *X <= x <= X[1])
    return*Y+(Y[1]-*Y)*(x-*X)/(X[1]-*X);
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
  template<class T>T CubeInterp(//<==========CUBIC (HERMITE SPLINE) INTERPOLATOR
      const T*X,//<--------------BRACKETING X VALUES (*X AND X[1] MUST BE VALID)
      const T*Y,//<--------------BRACKETING Y VALUES (*Y AND Y[1] MUST BE VALID)
      T x,//<---------------VALUE TO INTERPOLATE AT (TYPICALLY, *X <= x <= X[1])
      T m0,T m1){//<---------------------------------------SLOPES AT *X AND X[1]
    T t=(x-*X)/(X[1]-*X);
    return*Y+(x-*X)*(((m0+m1)*t-(2*m0+m1))*t+m0)+(Y[1]-*Y)*(3-2*t)*t*t;
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
  template<class T>T CardinalSlope(//<========CALCULATES SLOPES FOR CubeInterp()
      const T*X,//<-----------BRACKETING X VALUES (X[-1] AND X[1] MUST BE VALID)
      const T*Y,//<-----------BRACKETING Y VALUES (Y[-1] AND Y[1] MUST BE VALID)
      T t){//<---------------------------------------------------TENSION PARAMETER
    return(1-t)*(Y[1]-Y[-1])/(X[1]-X[-1]);
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
  template<class T>void PolyFit(//<=========FITS A POLYNOMIAL TO A SET OF POINTS
      const T*X,//<-------INDEPENDENT-VARIABLE VALUES (EACH X[i] MUST BE UNIQUE)
      const T*Y,//<-------------------DEPENDENT-VARIABLE VALUES (SAME SIZE AS X)
      int n,//<--------------------------------------NUMBER OF ELEMENTS IN X OR Y
      int m,//<----------------NUMBER OF COEFFICIENTS IN THE BEST-FIT POLYNOMIAL
      T*C){//<------STORAGE FOR COEFFICIENTS (SIZE=m) y=C[0]+C[1]*x+C[2]*x^2+...
    T**A=new T*[m];/*<-*/for(int i=0,j,k;i<m;++i){//...........augmented matrix
      for(A[i]=new T[m+1],j=0;j<m+1;++j)A[i][j]=0;
      for(k=0;k<n;++k)for(A[i][m]+=pow(X[k],i)*Y[k],j=0;j<m;++j)
        A[i][j]+=pow(X[k],i+j);}
    for(int i=1,j,k;i<m;++i)for(k=0;k<i;++k)for(j=m;j>=0;--j)//.........Gaussian
       A[i][j]-=A[i][k]*A[k][j]/A[k][k];//                           elimination
    for(int i=m-1,j;i>=0;--i)for(C[i]=A[i][m]/A[i][i],j=i+1;j<m;++j)//..backward
      C[i]-=C[j]*A[i][j]/A[i][i];//                                 substitution
    for(int i=0;i<m;++i)delete[]A[i];/*&*/delete[]A;
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
}//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
#endif
```

BinarySearch() Performance

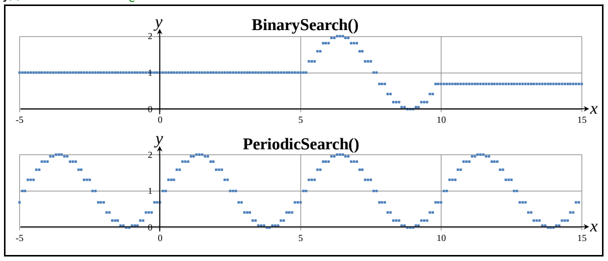
```
#include <algorithm>//.....................................................sort()
#include <cstdio>//......................................................printf()
#include <ctime>//.........................................clock(),CLOCKS_PER_SEC
#include "y_interp.h"//..................................................yInterp
#include "y_random.h"//..................................................yRandom
template<class T>T*LinearSearch(//<========FIND THE POINTER c | *c <= k < *(c+1)
    T*a,T*b,//<----ARRAY START & END POINTS (ARRAY MUST BE SORTED IN INC. ORDER)
    T k){//<-------------------------------------------------------------------KEY
  if(k<*a)return a-1;//...........note that a-1 may point to an invalid address
  while(k<*--b);/*->*/return b;
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
int main(){//<=========PERFORMANCE TEST FOR THE yInterp::BinarySearch() FUNCTION
  const int N=16384,M=10000000;//...max # of elements in array, # of repetitions
  unsigned I[625];/*<-*/yRandom::Initialize(I,1);//....state of Mersenne twister
  double*X=new double[N];/*<-*/for(int i=0;i<N;++i)X[i]=yRandom::RandU(I,0,N);
  std::sort(X,X+N);//........X is a sorted, unchanging list of numbers to search
  printf("          |     linear  search      |     binary  search\n");
  printf("    size  |-----------------------|-----------------------\n");
  printf("     of   |  search  |     index    |  search  |     index\n");
  printf("   array  | time (s) |    average   | time (s) |    average\n");
  printf(" ---------|----------|-------------|----------|-------------\n");
  for(int m=2;m<16385;m*=2){
    printf("%8d  |",m),yRandom::Initialize(I,1);
    double y=0,z=0,t=clock();
    for(int i=0;i<M;++i)y+=LinearSearch(X,X+m,yRandom::RandU(I,0,m))-X;
    printf("%8.3f  |%12.6f |",(clock()-t)/CLOCKS_PER_SEC,y/M),t=clock();
    yRandom::Initialize(I,1);
    for(int i=0;i<M;++i)z+=yInterp::BinarySearch(X,X+m,yRandom::RandU(I,0,m))-X;
    printf("%8.3f  |%12.6f\n",(clock()-t)/CLOCKS_PER_SEC,z/M);}
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```





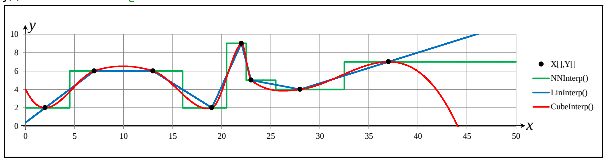
BinarySearch()/PeriodicSearch() Example

```
#include <cstdio>//..................................freopen(),printf(),stdout
#include "y_interp.h"//................................yInterp,<cmath>{sin()}
int main(){//<==============COMPARISON BETWEEN BinarySearch() & PeriodicSearch()
  freopen("sine.csv","w",stdout);
  double X[20];/*<-*/for(int i=0;i<20;++i)X[i]=5*i/20.+5;
  double Y[20];/*<-*/for(int i=0;i<20;++i)Y[i]=sin(2*3.14159*X[i]/5)+1;
  printf("x,y - BinarySearch()),y - PeriodicSearch()\n");
  for(double x=-5,k;x<15;x+=.1){
    printf("%f,%f,",x,Y[yInterp::BinarySearch(X+1,X+20,x)-X]);
    printf("%f\n",Y[yInterp::PeriodicSearch(X,X+20,k=x,5.)-X]);}
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```



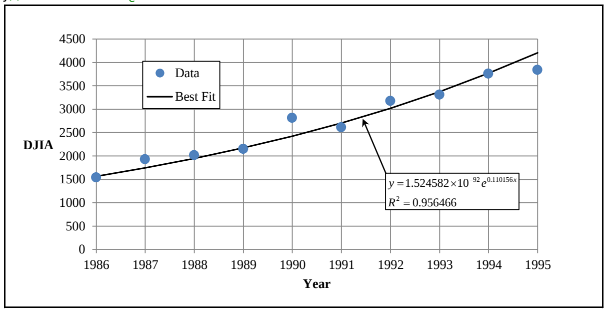
Interpolation Comparison

```
#include <cstdio>//..................................freopen(),printf(),stdout
#include "y_interp.h"//..................................................yInterp
int main(){//<================COMPARISON BETWEEN VARIOUS INTERPOLATION FUNCTIONS
  freopen("interp.csv","w",stdout);//..................redirect output to a file
  double X[8]={2,7,13,19,22,23,28,37},Y[8]={2,6,6,2,9,5,4,7};
  double m[8]={0};/*<-*/for(int i=1;i<7;++i)
    m[i]=yInterp::CardinalSlope(X+i,Y+i,0.);
  printf("#x,y,x,y1,y2,y3\n");
  for(int k=0,n=20000;k<n;++k){
    double x=50/(n-1.)*k;
    int i=yInterp::BinarySearch(X+1,X+8-1,x)-X;//..................note 0>=i<=6
    double y1=yInterp::NNInterp(X+i,Y+i,x);
    double y2=yInterp::LinInterp(X+i,Y+i,x);
    double y3=yInterp::CubeInterp(X+i,Y+i,x,m[i],m[i+1]);
    k<8?printf("%f,%f,",X[k],Y[k]):printf("-,-,");
    printf("%f,%f,%f,%f\n",x,y1,y2,y3);}
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```



PolyFit() Example

```
#include <cstdio>//..................................freopen(),printf(),stdout
#include <vector>//....................................................vector
#include "y_interp.h"//............................yInterp,<cmath>{exp(),pow()}
#include "y_io_2.h"//........................................................yIo
int main(){//<======FITTING AN EXPONENTIAL EQUATION USING THE PolyFit() FUNCTION
  //-------------------------READ AND PARSE FILE------------------------------
  char*s=yIo2::ReadTextFile("^dji_d.csv");
  std::vector<std::vector<char*> >S=yIo2::Parse2D(s,"/ \t-,");
  //----------------------INTERPRET DATES AND VALUES-------------------------
  std::vector<double>X,Y;
  for(int i=2,y;(y=atoi(S[i][0]))<1996;++i)
    if(y!=atoi(S[i-1][0])&&y>1984)X.push_back(y),Y.push_back(atof(S[i][6]));
  delete[]s;
  //-----------------------PERFORM EXPONENTIAL FIT---------------------------
  std::vector<double>Yp(X.size());
  for(int i=0,m=X.size();i<m;++i)Yp[i]=log(Y[i]);
  double*a=&X[0],*b=&Yp[0];
  double C[2];/*<-*/yInterp::PolyFit(a,b,X.size(),2,C);
  double y_bar=0;/*<-*/for(int k=0;k<10;++k)y_bar+=Y[k];/*&*/y_bar/=10;
  double SS_res=0,SS_tot=0;/*<-*/for(int k=0;k<10;++k)
    SS_res+=pow(Y[k]-exp(C[0]+C[1]*X[k]),2.),SS_tot+=pow(Y[k]-y_bar,2.);
  printf("A=%E\nB=%f\nR^2=%f\n",exp(C[0]),C[1],1-SS_res/SS_tot);
  freopen("exponential_fit.csv","w",stdout);
  for(int i=1,n=X.size();i<n;++i)
    printf("%f,%f,%f\n",X[i],Y[i],exp(C[0]+C[1]*X[i]));
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JUL2014~~~~~~
```



16. References

1. Yager RJ. Creating, Searching, and Deleting KD Trees Using C++. Aberdeen Proving Ground (MD): Army Research Laboratory (US). September 2014. Report No.: ARL-TN-0629.

2. Yager RJ. Working With Evenly Spaced, Rectangular Surface Grids Using C++. Aberdeen Proving Ground (MD): Army Research Laboratory (US). October 2014. Report No.: ARL-TN-0641.

3. Yager RJ. Generating Pseudorandom Numbers from Various Distributions Using C++. U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, June 2014. Report No.: ARL-TN-613.

4. Yager RJ. Reading, Writing, and Modifying BMP Files Using C++. Aberdeen Proving Ground (MD): Army Research Laboratory (US); August 2013. Report No.: ARL-TN-559.

5. Yager RJ. Reading, Writing, and Parsing Text Files Using C++ (Updated). Aberdeen Proving Ground (MD): Army Research Laboratory (US). October 2014. Report No.: ARL-TN-0642.

6. Lancaster P, Salkauskas K. Curve and Surface Fitting, an Introduction. San Diego (CA): Academic Press Ltd, 1987.

7. “^dji_d.csv”. [accessed 2014 Jun 19] http://stooq.com/q/d/?s=^dji.

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
2 (PDF)	DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL IMAL HRA MAIL & RECORDS MGMT
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
1 (PDF)	DIR USARL RDRL WML A R YAGER

INTENTIONALLY LEFT BLANK.